Assignment – 17

Gumma V L Prasad
(H.T.No: 2406CYS107)

1. Explain Data Encryption Standard (DES) and Rivest-Shamir-Adleman (RSA) Algorithms.

Answer:- Data Encryption Standard (DES)

The Data Encryption Standard (DES) is a **symmetric-key block cipher**, meaning it uses the same secret key for both encryption and decryption. It was a widely used encryption algorithm published in the 1970s by the National Institute of Standards and Technology (NIST). While no longer considered secure for most purposes due to its short key length, DES played a significant role in the development of modern cryptography.

Here's a breakdown of DES:

- **Block Size:** 64 bits (meaning it encrypts data in 64-bit chunks)
- **Key Size:** Technically 64 bits, but only 56 bits are effectively used for encryption (the remaining 8 bits are parity bits for error checking)
- **Rounds:** 16 (each data block goes through 16 rounds of encryption)
- **Structure:** Feistel Network (a specific design for block ciphers)

Limitations of DES:

- **Short Key Length:** With only 56 effective key bits, brute-force attacks (trying every possible key) became feasible with increased computing power.
- **Susceptibility to Attacks:** DES has known vulnerabilities, making it unsuitable for protecting sensitive data.

Rivest-Shamir-Adleman (RSA) Algorithm

Rivest-Shamir-Adleman (RSA) is an **asymmetric-key cryptography** algorithm, meaning it uses a different key for encryption and decryption (a public key and a private key). This makes it more secure than DES for certain applications.

Here's how RSA works:

- **Key Generation:** RSA involves complex mathematical operations based on large prime numbers. A public key and a private key are mathematically linked.
- **Encryption:** Data is encrypted using the public key, which is widely available. Only the private key can decrypt the data.

- **Digital Signatures:** RSA can also be used for digital signatures, allowing verification of the sender's identity and message integrity.

Advantages of RSA:

- **Stronger Security:** The key length in RSA is much larger than DES, making it significantly more resistant to brute-force attacks.
- **Digital Signatures:** Provides a mechanism for data authentication beyond encryption.

Disadvantages of RSA:

- **Computational Cost:** RSA is computationally expensive compared to DES, making it less efficient for large amounts of data.
- **Key Management:** Securely storing and managing the private key is crucial.

In summary:

- Use DES for historical context or educational purposes only.
- Use RSA for applications requiring strong security and digital signatures, but consider the computational cost.
- Modern encryption algorithms like AES (Advanced Encryption Standard) are preferred for most current applications due to their superior security and efficiency.

2. Explain Diffie-Hellman Key Exchange Algorithm With an Example.

Answer:- The Diffie-Hellman key exchange algorithm, developed by Whitfield Diffie and Martin Hellman, allows two parties (let's call them Alice and Bob) to establish a shared secret key for secure communication over an insecure public channel, without ever actually revealing the key itself! Here's how it works:

1. Agree on Public Values:

- Alice and Bob publicly agree on two large prime numbers:
  - **p (prime modulus):** A large prime number that acts as a base for the calculations.
  - **g (primitive root):** A number that, when repeatedly multiplied by itself modulo p, generates all the positive integers less than p (except for 0).

2. Private Key Generation:

- **Privately:** Alice chooses a random secret integer **a** (her private key).
- **Privately:** Bob chooses a random secret integer **b** (his private key).

3. Public Key Exchange:

- Alice calculates **A = g^a mod p** (a large number computed using her secret key a) and sends it publicly to Bob.
- Bob calculates **B = g^b mod p** (a large number computed using his secret key b) and sends it publicly to Alice.

4. Shared Secret Key Generation:

- **Alice:** Using Bob's public value (B) and her private key (a), Alice calculates the shared secret key: **K = B^a mod p**
- **Bob:** Using Alice's public value (A) and his private key (b), Bob calculates the same shared secret key: **K = A^b mod p**

Magic of Diffie-Hellman:

Even though an eavesdropper (Eve) can intercept the exchanged values (p, g, A, and B), she cannot calculate the secret key (K) because she doesn't know either Alice's private key (a) or Bob's private key (b). The mathematical properties of modular exponentiation make it easy to compute one-way (g^a mod p or g^b mod p) but incredibly difficult to reverse engineer and find the original secret key (a or b) from the public exchanged values.

Example:

Let's say:

- p (prime modulus) = 23
- g (primitive root) = 5
- Alice chooses a = 4 (her private key)
- Bob chooses b = 3 (his private key)

1. Alice calculates A = 5^4 mod 23 = 5 (public key) and sends it to Bob.
2. Bob calculates B = 5^3 mod 23 = 10 (public key) and sends it to Alice.
3. Alice calculates K = 10^4 mod 23 = 5 (shared secret key).
4. Bob calculates K = 5^3 mod 23 = 5 (shared secret key).

Even though Eve knows p, g, A, and B, she cannot find the secret key (K = 5) without knowing a or b.

This shared secret key (K) can now be used by Alice and Bob for secure communication over an insecure channel. They can encrypt messages using this key, ensuring only the intended recipient can decrypt them.

3. Explain Digital Signature Algorithm (DSA) With an Example.

Answer:- The Digital Signature Algorithm (DSA) is a public-key cryptography system used for digital signatures. Unlike Diffie-Hellman which establishes a shared

secret key, DSA allows verification of a message's authenticity and origin. Here's how it works:

Key Generation:

1. **Public Parameters:** A set of public parameters are chosen and shared:
    - **p (prime modulus):** A large prime number.
    - **q (subprime):** A prime number smaller than p.
    - **g (generator):** A number with a special property within the modular arithmetic of p and q.
2. **Private Key:** A random secret integer **x** (private key) is chosen by the signer (e.g., Alice).
3. **Public Key:** The public key **y** is computed as: **y = g^x mod p** and published for verification.

Signing a Message:

1. **Message Hash:** Alice creates a message (M) and calculates its hash (H(M)) using a secure hash function (like SHA-256).
2. **Random Value:** Alice chooses a random secret integer **k**.
3. **Signature Calculation:** Alice calculates two signature values:
    - **r = (g^k mod p) mod q**
    - **s = (k^-1 * (H(M) + x * r)) mod q**
4. **Signature (r, s):** The message signature is the pair (r, s) and is sent along with the message (M).

Verifying the Signature:

1. **Bob receives:** The message (M), signature (r, s), and Alice's public key (y).
2. **Verification Calculation:** Bob performs the following calculations:
    - **w = (s^-1) mod q**
    - **u1 = (H(M) * w) mod q**
    - **u2 = (r * w) mod q**
    - **v = ((g^u1 * y^u2) mod p) mod q**
3. **Verification:** If **v == r**, then the signature is valid, and the message likely originated from Alice (with her private key).

Example:

Let's say:

- p (prime modulus) = 101
- q (subprime) = 19
- g (generator) = 2
- Alice chooses x (private key) = 42
- Public key y = g^x mod p = 2^42 mod 101 = 13 (published)

Signing a message "Hello":

1. Hash(Hello) = 56 (using a secure hash function)
2. Alice chooses a random k = 78
3. Signature calculation:
    - r = (2^78 mod 101) mod 19 = 11
    - s = (78^-1 * (56 + 42 * 11)) mod 19 = 3
4. Signature sent: (Hello, 11, 3)

Verifying the Signature:

1. Bob receives the message "Hello", signature (11, 3), and Alice's public key (13).
2. Verification calculation:
    - w = 3^-1 mod 19 = 7
    - u1 = (56 * 7) mod 19 = 1
    - u2 = (11 * 7) mod 19 = 15
    - v = ((2^1 * 13^15) mod 101) mod 19 = 11
3. Verification: Since v (11) matches r (11), the signature is valid, and the message likely came from Alice.

Important Notes:

- DSA relies on the difficulty of solving the discrete logarithm problem, making it computationally expensive to forge a signature without the private key.
- DSA is less common in modern cryptography due to key size limitations compared to other algorithms like Elliptic Curve Digital Signature Algorithm (ECDSA).

4. Explain the Following Types of One-time Password (OTP) Algorithms with Examples:

a) Time-based OTP (TOTP)

b) HMAC-based OTP (HOTP)

Answer:- One-Time Password (OTP) Algorithms: TOTP vs HOTP

Both TOTP (Time-based OTP) and HOTP (HMAC-based OTP) are algorithms for generating one-time passwords used for two-factor authentication. However, they differ in how they calculate the OTP:

a. Time-based OTP (TOTP):

- **Concept:** TOTP uses the current time synchronized between the user's device (phone app) and the authentication server to generate the OTP.
- **Calculation:**
    1. A shared secret key is stored on both the server and the user's device.

2. The current time is divided into a specific time step (usually 30 or 60 seconds).
3. The current time step converted to a counter value (often represented in hexadecimal).
4. An HMAC (Hash-based Message Authentication Code) function is used to generate a one-time password using the shared secret key and the counter value.
5. The first few digits of the generated HMAC hash are extracted as the OTP (usually 6 or 8 digits).

Example:

- Shared secret key: "ThisIsASecretKey" (stored securely on server and user device)
- Current time: June 30, 2024, 15:55 IST (assuming 30-second time step)
- Counter value: Converted to hexadecimal (e.g., 0xFFFFFFF)
- HMAC generation: HMAC-SHA1("ThisIsASecretKey", 0xFFFFFFF) might result in a long hash value (e.g., "4F7…).
- OTP extraction: The first 6 digits (e.g., "4F7892") would be the one-time password valid for the current 30-second window.

b. HMAC-based OTP (HOTP):

- **Concept:** HOTP uses a counter value that increments with each generated OTP, independent of the current time.
- **Calculation:**
    1. A shared secret key is stored on both the server and the user's device.
    2. A counter value is maintained on the user's device (starts at 0 and increments for each OTP).
    3. An HMAC function is used to generate a one-time password using the shared secret key and the counter value.
    4. The first few digits of the generated HMAC hash are extracted as the OTP (similar to TOTP).

Example:

- Shared secret key: "ThisIsASecretKey" (stored securely on server and user device)
- Counter value on user device: 12345 (assuming the user has previously generated 12344 OTPs)
- HMAC generation: HMAC-SHA1("ThisIsASecretKey", 12345) might result in a long hash value (e.g., "DE3…).
- OTP extraction: The first 6 digits (e.g., "DE3F4G") would be the one-time password.

Key Differences:

- **Synchronization:** TOTP requires synchronized time between the server and user device, while HOTP does not.
- **Counter Management:** TOTP uses a counter derived from the current time, while HOTP maintains a separate counter that increments with each use.
- **Security:** Both are considered secure, but TOTP might be slightly less vulnerable to synchronization issues.

In Conclusion:

- TOTP is often preferred due to its time-based nature, reducing the risk of accidentally using an expired OTP.
- HOTP can be useful in situations where device time synchronization is difficult.